Integrate Mobile Synchronization with

Application Development Framework (ADF) Mobile Framework

Database Mobile Server 11g

Contact: Michael.Brey@Oracle.com

Oracle Database Mobile Server (DMS) provides data synchronization between client databases on mobile devices and an Oracle database back end. With Oracle Database Mobile Server, one can define data publications, mobile users, and user subscriptions to the publications. During the synchronization process, the data subscribed by the mobile user is downloaded and stored in the client database on the mobile device. As data is modified in the client database by the application, the new changes can be synchronized back to the server. This process is repeated many times over the lifetime of the application. See Chapter 2, in the *Oracle Database Mobile Server Developer's Guide* for more information about how synchronization works.

To run synchronization on a mobile device, you install a mobile client on the target platform and invoke the Open Sync Engine Synchronization Java API (OSE API) provided by DMS in the mobile application. A typical application modifies data in a client database and then synchronizes that data to an Oracle database back end.

This guide provides information on how to integrate DMS in an application developed with the ADF Mobile framework on an Android or iOS platform. On the Android platform, both Berkeley DB and SQLite are supported, however, Berkeley DB is recommended since it can support larger sized databases, extensible encryption capabilities and has a higher degree of concurrency which is needed in applications that run with DMS. For example, in most cases it is desirable to continue updating the data in the application while at the same time have a synchronization task running in the background. On the iOS platform, only SQLite is currently supported because everything is statically linked on that platform. Minor repackaging changes are needed before Berkeley DB can be swapped in as the client database on iOS.

The following sections describe the application development process using the OSE API in an ADF Mobile application.

1.1 So tware Preparation

This guide is not intended to be a replacement for the documentation provided by DMS and ADF Mobile. This guide shows one approach for connecting DMS into your application and it is possible there are other modifications or enhancements that can be done.

1.1.1 ADF Mobile Framework 11.1.2.4

Download and install Oracle JDeveloper and ADF Mobile extension:
http://www.oracle.com/technetwork/developer-tools/adf-mobile/downloads/index.html

- Setting Up ADF Mobile:
  http://docs.oracle.com/cd/E18941_01/tutorials/setupmobileapps/jdtut_11r2_54_1.html

- Oracle Fusion Middleware Mobile Developer's Guide for Oracle Application Development Framework: http://docs.oracle.com/cd/E37975_01/doc.111240/e24475/toc.htm

1.1.2 Oracle Database Mobile Server 11.3.0.0.0

Download Oracle Database Mobile Server 11.3.0.0.0:http://www.oracle.com/technetwork/products/database-mobile-server/downloads/index.html?ssSourceSiteId=ocomen

Note:

Follow the documents below and run the installer twice to install both DMS and the Mobile Development Kit.

- Document for installing Oracle Database Mobile Server 11.3.0.0.0:
  http://docs.oracle.com/cd/E48200_01/doc.1130/e38579/install.htm#BABEJCAH
- Document for installing Mobile Development Kit 11.3.0.0.0:
  http://docs.oracle.com/cd/E48200_01/doc.1130/e38579/install.htm#BABJGGBD

Note:

On supported platforms, install the Mobile Development Kit and JDeveloper on the same machine, as the Mobile Development Kit libraries are required for the application development in ADF Mobile.

1.1.3 Encryption Library

For either client database we need to add a java encryption library to handle credential encryption. The library we recommend is an open source project provided by Bouncy Castle. The standard JRE includes the Java Cryptography Extension (JCE) framework and implementation; however, the JRE used in ADF Mobile does not include the full implementation of JCE.

1.2 Con igure Your Application to Run on Android

The configuration is different for Berkeley DB and SQLite. Steps for configuring Berkeley DB are described in section **1.2.1 Be keley DB** while steps for SQLite are described in section **1.2.2 SQLite**.

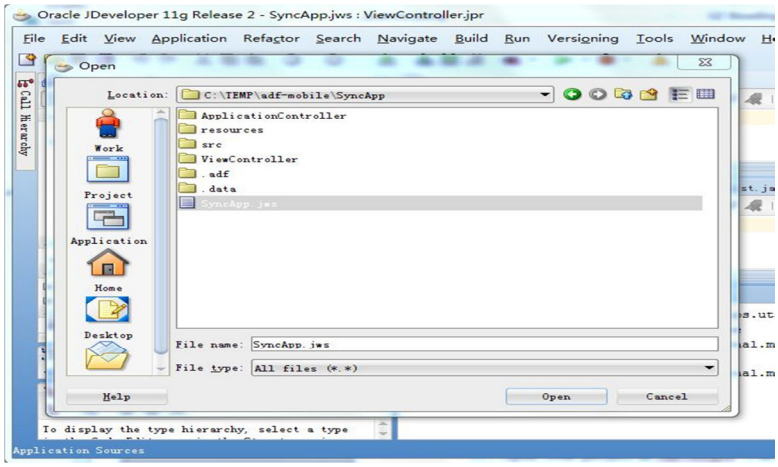

Figure 1 - Open Project

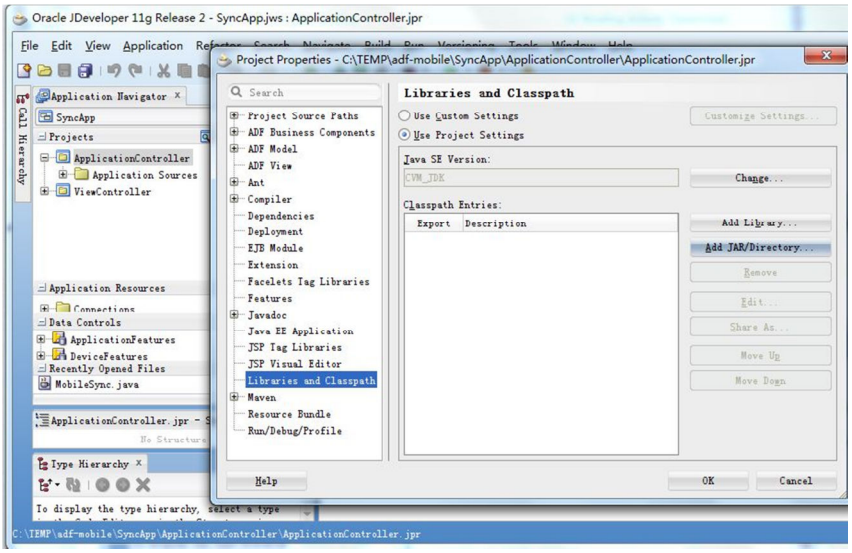

Figure 2 - Add Jars

1.2.1 Berkeley DB

To run sync with a Berkeley DB client, follow the steps to configure your project:

1) Open your project in JDeveloper.

- Choose *File → Open*.
- Choose your project file, click *Open*. See **Figure 1 - Open Project**.

2) Add libraries into Classpath:

- Right click *ApplicationController* in *Projects* tab, and choose *Libraries and Classpath*.
- Click *Add Jar/Directory*.

- Add two jars: sync library *osync_me.jar*, Berkeley DB JDBC driver *sqlite.jar*. See **Fig re 2 - Add Jars**. The sync library *osync_me.jar* is referred to as the OJEC client. The Android client is not compatible with ADF Mobile and should not be used.
- Copy *osync_me.jar* from *MOBILE_SDK_HOME\mobile\sdk\j2me\ojec* directory, where *MOBILE_SDK_HOME* is the directory where you installed the mobile development kit.
- Copy *sqlite.jar* from *MOBILE_SDK_HOME\mobile\sdk\j2me\bdb* directory, where *MOBILE_SDK_HOME* is the directory where you installed the mobile development kit.

3) Make changes to Berkeley DB sync to enable the android deployment template in *JDEVELOPER_HOME\jdev\extensions\oracle.adf.mobile\Android\Oracle_ADFmf_Framework.zip*.

The Android deployment template is part of the ADF Mobile framework that provides base framework services and code for the mobile application. Subsequent ADF Mobile applications built with the changed template will be Berkeley DB sync enabled.

- Replace *Oracle_ADFmf_Framework.zip\framework\Android\build\java_res_debug\assets\storage\jvm\lib\sqlite.jar* with Berkeley DB JDBC driver *sqlite.jar*
- Add Berkeley DB JDBC driver jni *liboracle-jdbc.so* to *Oracle_ADFmf_Framework.zip\framework\Android\build\java_res_debug\assets\storage\jvm\lib*

  Note:

  You can get *liboracle-jdbc.so* from *MOBILE_SDK_HOME\mobile\sdk\j2me\bdb\liboracle-jdbc.so*, where *MOBILE_SDK_HOME* is the directory where you installed mobile development kit.

- Add *bcprov-jdk14-146.jar* and *jce.jar* to *Oracle_ADFmf_Framework.zip\framework\Android\build\java_res_debug\assets\storage\jvm\lib\ext*
  - Download *bcprov-jdk14-***.jar* from http://www.bouncycastle.org/latest_releases.html
  - Copy *jce.jar* from *JDEVELOPER_HOME\jdev\extensions\oracle.adf.mobile\lib\jce.jar*
- Edit *Oracle_ADFmf_Framework.zip\framework\Android\build\java_res_debug\assets\storage\jvm\lib\security\java.security,* add the following:

  security.provider.[n]=org.bouncycastle.jce.provider.BouncyCastleProvider

  Where *n* is a sequential number following the last number in the list of the existing providers already included in the file.

Note:

In this step, you only change the debug package of the template in the *Oracle_ADFmf_Framework.zip\framework\Android\build\java_res_debug* directory. You can change the release

package of the template in the *Oracle_ADFmf_Framework.zip\framework\Android\build\java_res_release* directory as well if the application runs in release mode..

4) Create a deployment profile by choosing *Application → Deploy → New Deployment Profile*. By default, it will be in debug mode. For information on steps to create a deployment profile, see the section 17.2.1 and section 17.2.3 of the *Oracle Fusion Middleware Mobile Developer's Guide for Oracle Application Development Framework*.

1.2.2 SQLite

To run the application with SQLite client, follow the steps to configure the application:

1) Open your project in JDeveloper:

- Choose *File → Open*.
- Choose your project file, click *Open*. See Error! Reference source not found.**Fig re 1 - Open Project.**

2) Add libraries into Classpath:

- Right click *ApplicationController* in *Projects* tab.
- Choose *Libraries and Classpath*.
- Click *Add Jar/Directory*.
- Add 1 Jar: sync library *osync_me.jar*. See **Fig re 2 - Add Jars**. The sync library *osync_me.jar* is referred to as the OJEC client. The Android client is not compatible with ADF Mobile and should not be used.
- Copy *osync_me.jar* from *MOBILE_SDK_HOME\mobile\sdk\j2me\ojec* directory, where MOBILE_SDK_HOME is the directory where you installed mobile development kit.

3) Make changes to SQLite sync enable the Android deployment template in *JDEVELOPER_HOME\jdev\extensions\oracle.adf.mobile\Android\Oracle_ADFmf_Framework.zip*. The Android deployment template is part of the ADF Mobile framework that provides base framework services and code for the mobile application. Subsequent ADF Mobile applications built with the changed template will be SQLite sync enabled.

- Add *bcprov-jdk14-146.jar* and *jce.jar* to *Oracle_ADFmf_Framework.zip\framework\Android\build\java_res_debug\assets\storage\jvm\lib\ext*
  - Download *bcprov-jdk14-***.jar* from http://www.bouncycastle.org/latest_releases.html
  - Copy *jce.jar* from *JDEVELOPER_HOME\jdev\extensions\oracle.adf.mobile\lib\jce.jar*
- Edit *Oracle_ADFmf_Framework.zip\framework\Android\build\java_res_debug\assets\storage\jvm\lib\security\java.security*, add the following

security.provider.[n]=org.bouncycastle.jce.provider.BouncyCastleProvider

Where *n* is a sequential number following the last number in the list of the existing providers already included in the file.

Note:

In this step, you only change the debug package of the template in the *Oracle_ADFmf_Framework.zip\framework\Android\build\java_res_debug* directory. You can change the release package of the template in the *Oracle_ADFmf_Framework.zip\framework\Android\build\java_res_release* directory as well if the application runs in release mode.

4) Create a deployment profile by choosing *Application → Deploy → New Deployment Profile*. By default, it will be debug mode. For information on steps to create a deployment profile, see the section 17.2.1 and section 17.2.3 of the *Oracle Fusion Middleware Mobile Developer's Guide for Oracle Application Development Framework*.

5) On the version of ADF Mobile tested, we found that the SQLite libraries that come with ADF Mobile do not support the creation of statement journal files. These files are used by SQLite to rollback partial results of a single statement within a larger transaction. The version of SQLite that comes bundled with Android allows for the temporary journal files to be stored in memory. The DMS synchronization process uses these temporary journal files. The workaround is to replace ADF Mobile's prebuilt SQLite libraries with Android's SQLite libraries located in /system/lib before calling the first synchronization so that sync can use Android's libraries to create the sqlite databases. Replace the libsqlite_jni.so and libsqlite.so libraries in JDEVELOPER_HOME\jdev\extensions\oracle.adf.mobile\Andoird\Oracle_ADFmf_Framework.zip\framework\Android\build\java_res_debug\assets\storage\jvm\lib with the Android built libraries.

1.3 Con igure Your Application to Run on iOS

Use the following steps to configure the application to run on iOS with SQLite:

1) Open the project in JDeveloper.

- Choose *File → Open*
- Choose your project file, click *Open.* See **Fig re 1 - Open Project**.

2) Add libraries into Classpath:

- Right click *ApplicationController* in *Projects* tab.
- *Choose Libraries and Classpath.*
- *Click Add Jar/Directo*ry.
- Add 1 Jar: sync library *osync_me.jar*. See **Fig re 2 - Add Jars**. The sync library *osync_me.jar* is referred to as the OJEC client. The iOS native client is not compatible with ADF Mobile and should not be used
- Copy *osync_me.jar* from *MOBILE_SDK_HOME\mobile\sdk\j2me\ojec* d*irectory*, where *MOBILE_SDK_HOME* is the directory where you installed mobile development kit.

3) Make changes to SQLite sync and enable the iOS deployment template in *JDEVELOPER_HOME\jdev\extensions\oracle.adf.mobile\iOS.* As before, subsequent ADF Mobile applications built with the changed template will be SQLite sync enabled.

- Add both *bcprov-jdk14-146.jar* and *jce.jar* to the directory -

JDEVELOPER_HOME\jdev\extensions\oracle.adf.mobile\iOS\jvmti\x86\ext if the application is running on a simulator, or

JDEVELOPER_HOME\jdev\extensions\oracle.adf.mobile\iOS\jvmti\arm\ext if the application is running on a physical device.

  - Download *bcprov-jdk14-***.jar* from http://www.bouncycastle.org/latest_releases.html
  - Copy *jce.jar* from *JDEVELOPER_HOME\jdev\extensions\oracle.adf.mobile\lib\jce.jar*

- Edit

JDEVELOPER_HOME\jdev\extensions\oracle.adf.mobile\iOS\jvmti\x86\security\java.security, if the application is running on a simulator, or

JDEVELOPER_HOME\jdev\extensions\oracle.adf.mobile\iOS\jvmti\arm\security\java.security, if the application is running on a physical device.

- Add the following

```
security.provider.[n]=org.bouncycastle.jce.provider.BouncyCastleProvider
```

Where *n* is a sequential number following the last number in the list of the existing providers already included in the file.

4) Create a deployment profile by choosing *Application → Deploy → New Deployment Profile*. By default, it will be in debug mode. For information on steps to create a deployment profile, see the see the section 17.2.1 and section 17.2.3 in the *Oracle Fusion Middleware Mobile Developer's Guide for Oracle Application Development Framework*.

1.4 Implement Synchronization Functionality

DMS must be set up to allow mobile applications to synchronize (sync for short) with an Oracle database back end. See section **1.4.1 Set Up Database Mobile Se ve** , for more information.

After setting up the mobile server, mobile applications can invoke the OSE API with predefined credentials to synchronize. Mobile applications need to create the client database on the device by invoking the OSE API and then doing some modifications in client databases. When this is completed for the first time on the device, client databases are created in the *SQLITE.DATA_DIRECTORY/USERNAME/* directory, where *SQLITE.DATA_DIRECTORY* is a parameter in the mobile client configuration file

ose.ini and *USERNAME* is the user name used to perform the sync. The directory location can be changed by changing the value of *SQLITE.DATA_DIRECTORY*.

The client database name is specified when creating the publication (as described in step 1) in section **1.4.1 Set Up Database Mobile Se ve** . Once the publication is created, the corresponding client database name cannot be changed on the mobile device. Usually, the client database name is the same as the publication name. See sections **1.4.2 Ove view of Implementing Synch onization Functionality**, **1.4.3 Invoke Sync**, and **1.4.4 Get Sync Agent Status and Cont ol Sync Agent** for information on implementing the sync functionality.

1.4.1 Set Up Database Mobile Server

Perform the following to set up DMS before any synchronization:

1) Create a publication. See sections 4.1-4.8 in the *Oracle Database Mobile Server Developer's Guide* for steps on how to create a publication using the mobile development kit.
2) Package the publication and publish it. See section 5.1 in the *Oracle Database Mobile Server Developer's Guide* for steps on how to package the publication and publish it using the packaging wizard.
3) Create a user. See section 4.3.1.2 in the *Oracle Database Mobile Server Administration and Deployment Guide* for steps on how to create a user on mobile manager.
4) Subscribe the user to the application. See section 4.4.1.1 in the *Oracle Database Mobile Server Administration and Deployment Guide* for steps on how to grant application access to a user.
5) Start up the mobile server. See section 4.5 in the *Oracle Database Mobile Server Installation Guide* for information on how to start the mobile server.

Note:

Once these steps are completed, the mobile server is ready for synchronization with the mobile application.

1.4.2 Overview o Implementing Synchronization Functionality

DMS provides an OSE API to initiate tasks and maintain the sync process, for example starting/stopping/pausing/resuming sync, determining the sync status, enabling/disabling sync capability, etc. See section 3.1.1.1 in the *Oracle Database Mobile Server Developer's Guide* for more information about the OSE API.

There are two ways to add synchronization functionality into the mobile application, namely, setting up appropriate UIs to allow manual user control or letting the application handle it implicitly (automatic). For the manual user control approach, it is recommended to use one UI screen to enter the sync credentials and another one to control/manage the sync operations. This is considered as a feature approach.

If the application handles it automatically, then set up application-level preferences used for synchronization and invoke the sync method in the OSE API. With this approach, the application can sync data after each insert/delete/update statement and the end user would be unaware.

To implement synchronization functionality as a feature, perform the following steps:

1) Follow Chapter 6 of the *Oracle Fusion Middleware Mobile Developer's Guide for Oracle Application Development Framework* to create a feature, a task flow for this feature and two AMX pages - one for sync as shown in **Fig re 3 - Mobile Sync** and the other for sync agent as shown in **Error! Reference source not found.**. The figures are part of the HR sample application and are shown here as a sample layout. The AMX page for sync as shown in **Fig re 3 - Mobile Sync** provides a UI to enter user name, password and server URL, which are used to perform the sync. It also provides two buttons - one to invoke sync and the other to navigate to the AMX page for sync agent as shown in **Fig re 4 - Sync Agent**.

   The AMX page for sync agent as shown in **Fig re 4 - Sync Agent** provides buttons to control sync agent, such as start/stop/resume/pause/disable/enable and also shows sync agent status information as well as automatic sync information.

2) Implement sync functionality in Java code using the OSE API, which is provided in *osync_me.jar*. See section **1.4.3 Invoke Sync** to implement the sync functionality with OSE API.
3) To expose the sync functionality in Java code, you need to create Java Bean Data Controls in the ADF Mobile application and then add the functionality to the AMX pages to execute sync operations.

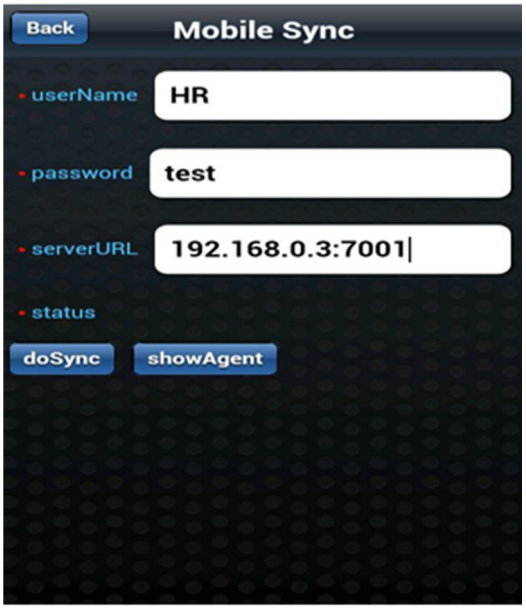

Figure 3 - Mobile Sync

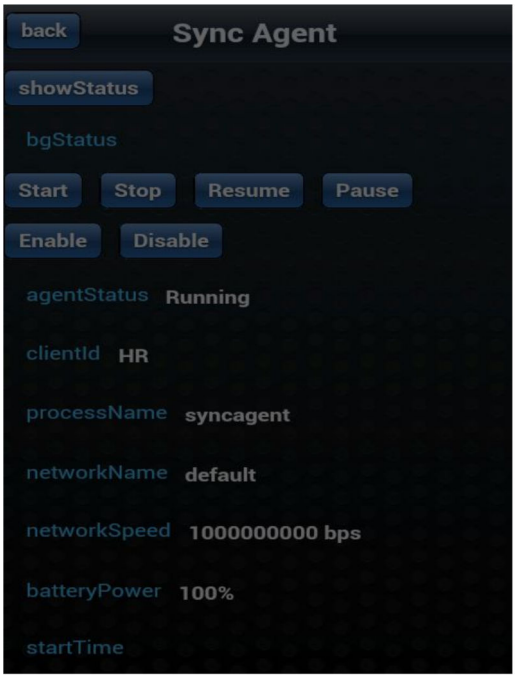

Figure 4 - Sync Agent

See instructions in Part IV in the *Oracle Fusion Middleware Mobile Developer's Guide for Oracle Application Development Framework* to finish steps 1 and 3 mentioned above.

To implement synchronization functionality implicitly, follow the steps below:

a) See Chapter 13 in the *Oracle Fusion Middleware Mobile Developer's Guide for Oracle Application Development Framework* to create an application-level preferences for username, password, server URL. The preferences can be referenced using the *Expression Language* added to AMX pages or via the ADF Mobile Java API *AdfmfJavaUtilities.evaluateELExpression(String expression)*.
b) Set parameters in ose.ini and call a manual sync using the OSESession API as described in section **1.4.3 Invoke Sync**.
c) Control automatic sync functionality in Java code using BGSession API, which is provided in *osync_me.jar*. Invoke automatic sync functionality in the application at the appropriate location, for instance, do sync after one insert statement is executed. See the example below in section **1.4.4 Get Sync Agent Status and Cont ol Sync Agent** for more information on what BGSession API to use.

1.4.3 Invoke Sync

DMS provides two kinds of synchronization, namely, manual synchronization and automatic synchronization.

Manual synchronization is initiated by invoking the OSE API in mobile applications. *OSESession* is a class in that API which exposes methods to invoke and control manual synchronization by setting synchronization parameters and options. For more information on manual synchronization see section

2.1.3 in the *Oracle Database Mobile Server Developer's Guide* and for information on *OSESession,* see section 3.1.1.1.2 in the *Oracle Database Mobile Server Developer's Guide*.

Automatic synchronization is managed by the Sync Agent component of the sync client which runs in the background and triggers sync under specific events and conditions. There are five types of events and conditions that are monitored by the sync agent: client data DMLs (SQL rules), time, device's network, power, and available memory. For example, one can define a publication data rule that triggers sync after every time 10 or more records in a database are modified, or schedule a periodic sync every 15 minutes. The conditions for the automatic sync could include minimum network speed and minimum battery level. User applications can control the Sync Agent as well as get the status of the monitored properties through the BGSession API. For more information on automatic synchronization, see section 2.1.3 in the *Oracle Database Mobile Server Developer's Guide* and on *BGSession,* see section 3.2.1.1.2 in the *Oracle Database Mobile Server Developer's Guide*.

This section describes how to invoke the OSE API in mobile applications to initiate manual synchronization while section **1.4.4 Get Sync Agent Status and Cont ol Sync Agent** describes how to invoke the sync agent control API in mobile applications to manage automatic synchronization.

Before invoking manual sync, several parameters need to be set in the *ose.ini* file. This file is a client-side configuration file and it can be generated and modified using *SetParam.run ().* The *ose.ini* file is located in the directory where *osync_me.jar* is located.

SQLITE.DATA_DIRECTORY is a parameter used to specify the location of the client database. For the example below, *SQLITE.DATA_DIRECTORY* can be set to ADF Mobile application directory. After the first sync, the client database is located in the *SQLITE.DATA_DIRECTORY/USERNAME/* directory, where USERNAME is the user name used to do the sync. It is important to specify the correct path name (directory) to the client database. Without it, the sync agent will not be able to find the database and it will be impossible to make data changes in the client database. See section 5.2 in *Oracle Database Mobile Server Mobile Client Guide* for more information on *ose.ini* and *SetParam.*

Sample Java code for manual synchronization is given as follows:

```
String dir = AdfmfJavaUtilities.getDirectoryPathRoot(AdfmfJavaUtilities.ApplicationDirectory);

String params[] = new String[4];

//disableAgent is a string variable defined to either YES or NO

params[0] = "BGSYNC.DISABLE=" + disableAgent;

params[1] = "NETWORK.DISABLE_SSL_CHECK=YES";

params[2] = "OSE.FILES=YES";

params[3] = "SQLITE.DATA_DIRECTORY=" + dir;

// Set sync parameters in ose.ini before doing sync

SetParam.run(params);
```

```
// The following 4 lines of code are only needed if you are using the BDB client database. They
are used to set up BDB's JDBC driver.  If you are using the SQLite client database, then you can
comment out those 4 lines. The additional params are added to ose.ini.

String bdbParams[] = new String[2];

bdbParams[0] = "SQLITE.JDBC.URL_PFX=jdbc:sqlite:/";

bdbParams[1] = "SQLITE.JDBC.DRIVER=SQLite.JDBC";

SetParam.run(bdbParams);

// Initialize OSESession with username and password.

OSESession sess = new OSESession(userName, password.toCharArray());

// Set mobile server URL.

sess.setURL(serverURL);

// Save user information.

sess.saveUser();

// Set save password to true so that sync agent can get password for background sync.

sess.setSavePassword(true);

// Do a sync.

sess.sync();
```

1.4.4 Get Sync Agent Status and Control Sync Agent

Automatic synchronization is enabled by default if a publication is enabled for automatic synchronization. It can be managed by invoking the sync agent control API which is exposed in the *BGSession* class. You can get sync agent status or start/stop/resume/pause/enable/disable the agent using the methods in *BGSession*. When using automatic synchronization, the first sync must be a manual one because this will create the client databases on the mobile device. See section **1.4.3 Invoke Sync** for more information on manual sync. Several parameters need to be set in ose.ini (BGSYNC.DISABLE set to NO, username/password saved, etc.). The following is the sample code:

```
// Initialize an instance of BGSession in constructor

public BgSync() {

    super();
```

```
        try {
            if(mBgSess == null)
            mBgSess = new BGSession();
        }catch(Exception e) {
        e.printStackTrace();
        }
    }
```

You can start/stop/resume/pause/enable/disable the agent using methods in *BGSession*. For example,

```
// Start sync agent
mBgSess.start();
```

See section 3.2.1.1.2 in the *Oracle Database Mobile Sever Developer's Guide* for more information.

```
// Show sync agent status, including running status, network speed and so on.
BGAgentStatus as = mBgSess.getAgentStatus();
```

You can use *BGAgentStatus* class to get the sync user name, the name of the application or process that is executing the sync agent, remaining percentage of battery life, name of the network currently used for sync, network speed, and status of the sync agent. For example, you can use *as.statusCode* to get the status of the sync agent. See section 3.2.1.1.3 in the *Oracle Database Mobile Server Developer's Guide* for more information.

```
// Show status information on automatic sync.
BGSyncStatus ss = mBgSess.getSyncStatus();
```

You can use *BGSyncStatus* class to get the status of automatic sync, including start time of current or last sync, end time of last sync, priority of the current or last sync, exception object thrown during last sync and so on. For example, you can use *ss.startTime* to get the start time of the current or last sync. See section 3.2.1.1.4 in the *Oracle Database Mobile Server Developer's Guide* for more information.

After implementing the sync functionality, run the application using *Application → Deploy → YOUR_DEPLOYMENT_PROFILE*. For information on the steps to deploy an application, see section 17.3 of the *Oracle Fusion Middleware Mobile Developer's Guide for Oracle Application Development Framework*.